APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **DYNAMIC OBJECT USAGE PATTERN LEARNING AND EFFICIENT CACHING**

APPLICANT: **Yury Kamen, Robert N. Goldberg, Bruce K. Daniels, Syed M. Ali, and Peter A. Yared**

"EXPRESS MAIL" Mailing Label Number: EL656799693US

Date of Deposit: November 30, 2001

DYNAMIC OBJECT USAGE PATTERN LEARNING AND EFFICIENT CACHING

Cross-Reference to Related Applications

[0001] This application relates to U.S. Patent Application Serial No. _____, entitled "Transparent Injection of Intelligent Proxies into Existing Distributed Applications," filed contemporaneously herewith.

Background of Invention

Field of the Invention

[0002] The present invention relates generally to methods for optimizing performance of distributed applications. More particularly, the present invention relates to a method for optimizing performance of a distributed application by dynamic usage pattern learning.

Background Art

[0003] Many distributed applications use object caches of some kind to reduce data roundtrips between the client and the server. In order for the object caching to be beneficial, the caching policy, *i.e.*, which objects are replaced when the cache is full, which are objects are pre-fetched into the cache, and what object attributes are pre-fetched, must be efficient. It is common to find excessive amounts of data being transferred over the network and cached on the client because of overzealous data pre-fetching in false anticipation of future usage of the data. For example, if a Business Manager object has a list of immediate reports as an attribute and the naïve pre-fetching algorithm gets them all along with the Manager object, and does the same for each report. Then if there is a request for data on the CEO of a large corporation, naïve pre-fetching will result in attempting to cache every employee's profile. It is also common to find excessive number of data roundtrips between the client and the server because of indiscriminate use of fine-grained remote objects. The amount of data transferred and the

number of data roundtrips between the client and the server may be reduced if pre-fetching is based on data usage patterns, since many distributed applications follow a small number of data usage patterns (relative to the amount of the data itself) repeated over many times. For example, data usage patterns in most enterprise applications are determined by business logic rules and data relationships, which are typically very small in comparison to the amount of the application data.

[0004] One of the main challenges facing many enterprises is how to enable their legacy applications to make use of more efficient techniques, such as object pre-fetching based on dynamic usage pattern learning, without having to redesign the applications or while migrating to a new architecture. Many legacy applications are designed without intent or means to facilitate learning of data usage patterns. Further, there is limited or no means for application source static analysis and re-architecture due to legal restrictions on accessing the application source code and/or the application relying on third party libraries and complex interactions with the distributed environment. Nevertheless, the effect of pre-fetching objects and attributes based on dynamic usage pattern learning can be quite beneficial to the performance of the application. Therefore, what is desired is a method for learning usage patterns dynamically in distributed applications designed without intent or means to facilitate learning of data usage patterns.

Summary of Invention

[0005] In one aspect, the invention relates to a method for transparently optimizing data access. The method comprises gathering information related to data usage when a system is processing and generating a usage pattern of the system using gathered information.

[0006] In one aspect, the invention relates to a method for transparently optimizing data access. The method comprises gathering information related to data usage when a system is processing, generating a usage pattern of the system using gathered information, pre-fetching data determined by the usage pattern of the system, caching data locally, accessing data by the system, and synchronizing cached data with persistent data.

[0007] In one aspect, the invention relates to a method for transparently optimizing a distributed application having a client-side and a server-side application. The method comprises gathering information related to data usage on the client-side when the distributed application is processing, and determining a usage pattern using gathered information.

[0008] In one aspect, the invention relates to a method for transparently optimizing a distributed application having a client-side and a server-side application. The method comprises gathering information related to data usage on the client-side when the distributed application is processing, determining a usage pattern using gathered information, pre-fetching data from the server-side, caching data on the client-side, accessing data on the client-side, and synchronizing cached data on the client-side with persistent data on the server-side.

[0009] In one aspect, the invention relates to a computer-readable medium having recorded thereon instructions executable by processing. There are instructions for gathering information related to data usage when a system is processing, determining a usage pattern of the system using gathered information, pre-fetching data determined by the usage pattern of the system, caching data locally, accessing data by the system, and synchronizing cached data with persistent data.

[0010] In one aspect, the invention relates to an apparatus for transparently optimizing data access. The apparatus comprises means for gathering information related to data usage when a system is processing, means for generating a usage pattern of the system using gathered information, means for pre-fetching data determined by the usage pattern of the system, means for caching data locally, accessing data by the system, and means for synchronizing cached data with persistent data.

[0011] Other features and advantages of the present invention will be apparent from the following description and the appended claims.

Brief Description of Drawings

[0012] Figure 1 is a block diagram of a distributed application in which a client application accesses application data in a data store through a server application.

[0013] Figure 2 is an exemplary set of objects, some of which include methods, to be accessed by the client application of Figure 1.

[0014] Figure 3 shows an optimized version of the distributed application of Figure 1 in accordance with one embodiment of the present invention.

Detailed Description

[0015] The present invention relates to a method for learning data usage patterns in a system that was designed/developed without the intent or means to facilitate learning of data usage patterns. Database applications, file server applications, etc. are examples of such systems. Further, the present invention provides a method of analyzing and instrumenting the system to intercept and monitor operations related to system components. Further the present invention relates to a method of deriving a usage pattern based on intercepting and monitoring components. Further, the present invention provides a method of using the usage pattern to optimize data transfer between different portions of the system.

[0016] Embodiments of the present invention provide a method for learning data usage patterns in existing distributed applications that are designed/developed without intent or means to facilitate learning of data usage patterns. The method of the present invention involves analyzing and instrumenting the distributed application to intercept and monitor operations related to objects on the client-side of the application. Such operations include searching for objects, creating and destroying objects, accessing attributes of objects, and modifying objects via remote calls. When the instrumented application is running, the usage pattern of the objects is derived by monitoring the client-side objects and collecting execution-related information, such as the attributes of the objects accessed and the modifications made to the objects via remote method calls. This usage pattern is used to determine a set of objects to pre-fetch and cache on the client and the attributes to preload

in the objects. The optimal amount of data can be sent between the client and server by determining the cached data changed by the user and the server data changed as a result of executing a method on the server. The usage pattern learning can start from an initial access pattern based on analysis of the application code or from an empty set.

[0017] In other to fully understand the present invention, it is useful to first consider an existing distributed application that does not include a facility for learning data usage patterns and then show how the present invention enables the application to learn data usage. In the following discussion, numerous specific details are set forth to facilitate understanding of the present invention. However, it will be apparent to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail to avoid obscuring the present invention.

[0018] Figure 1 shows an example of a distributed system **5** that can be optimized by the present invention. For illustration purposes, it is assumed that the distributed system **5** does not include a facility for learning data usage patterns. The distributed system **5** includes a web server **15** hosting web components **10**, such as JavaServer Pages (JSP) pages and Java servlets. The web components **10** contain the presentation logic that defines what a web client **20**, typically a web browser, displays and how requests from the web client **20** are handled. The web client **20** and the web server **15** communicate over a network link **25**. The distributed system **5** further includes an application server **35** having persistent objects **40**, such as Enterprise JavaBeans (EJB) entities, that model the data held within a data store **45**. The application server **35** communicates with the web components **10** over a network link **50**. The persistent objects **40** can have numerous attributes and numerous methods that can be invoked by the web components **10** in order to manipulate data stored within the data store **45**.

[0019] Figure 2 shows a representative collection of the persistent objects (**40** in Figure 1) in the application server (**35** in Figure 1). The collection of objects includes an Order object **40a**, a Customer object **40b**, a Line Item object **40c**, a Product object **40d**, and an Employee object **40e**. The Order object **40a** represents a sales order. The Customer

object **40b** holds information regarding the customer that placed the order, such as first name, last name, billing address, and credit card number. The Line Item object **40c** identifies particular items in the sales order. The Product object **40d** holds information regarding the ordered item. The Employee object **40e** holds information regarding the employee that accepted the order. In the figure, the relationships between the objects are represented by arrows. Each of the objects **40a**, **40b**, **40c**, **40d**, **40e** can have numerous attributes and methods. As an example, the Order object **40a** has three attributes: id, date, and status. The Order object **40a** also includes a method `getOrderTotal()`, which retrieves the total cost of the order.

[0020] Returning to Figure 1, the web components **10** use remote calls to access and invoke methods on the persistent objects **40**. As an example, if the persistent object **40** is an EJB entity, web component **15** first acquires a home interface for the EJB entity from a lookup service **55**. The web component **15** then uses the home interface to create or find an instance of the EJB entity. The web component **15** can use the home interface to execute methods on the EJB entity instance, get a reference to the EJB entity instance, and remove the EJB entity instance. The distributed system **5** includes a cache **60** on the web server **15** for holding the accessed on the application server **35**. The objects are cached as proxy objects that can be locally accessed by the web components **10**.

[0021] In operation, pages are displayed on the web client **20** using selected attributes from selected objects. For example, one page may display names of customers only (using Customer object) and allow a user to select a button to display orders associated with a specific customer. Another page may display orders associated with a specific customer (using Order and Customer objects) and allow a user to select a button to display details about a specific order. Another page may display details about a specific order (using Order, Line Item, Product, Customer, and Employee objects) and allow the user to make changes to the order. In order to generate each of these pages, the required objects must either be present in the client cache **60** or fetched from the application server **35**. If the object usage pattern is known, then the optimal amount of data can be pre-fetched from the application server **35** before each page is generated. The object

usage pattern would include information about what objects and attributes are needed to generate each page.

[0022] In accordance with one embodiment of the present invention, the distributed system **5** is instrumented to intercept calls to the lookup service **55**, calls to create, find, and destroy object instances, calls to access attributes of objects, and remote calls to modify objects. Figure 3 shows the distributed system **5** after instrumentation. As shown, a client runtime **65** is interposed between the web components **10** and the application server **35** to intercept all the calls listed above in order to gather information about object usage. The web components **10** are modified such that calls that were originally made to the application server **35** and lookup service **55** now go to the client runtime **65**. Typically, the process of modifying the web components **10** involves parsing the machine code (bytecode) or source code for the web components **10** and replacing all the calls listed above with calls to the client runtime **65**. So, for example, calls to the lookup service **55** are replaced with calls to a lookup service included in the client runtime **65**. This allows the client runtime **65** to collect information about the object that the web component is trying to locate.

[0023] When the instrumented application is running, the application transitions from one state to another in response to user gestures. As the application transitions from one state to another, the client runtime **65** monitors object usage on the client-side, collects execution related information, and derives object usage patterns. The object usage patterns are then used to determine a set of objects and attributes to pre-fetch and store in the client cache **60**. The objects and attributes can be pre-fetched based on a number of criteria. For example, the decision can be based on application-dependent positions such as the URL of web server pages or application stack trace at points where objects are created and/or modified and/or destroyed. The decision can also be based on object type, object attribute access statistics, size of object or its individual attributes. Other criteria include similarity in usage patterns of attributes of the same or different objects, size of the data sent between the client and the server in each roundtrip or a set of roundtrips within a time window, the structure of the object graph sent from the server to the client,

the place of the object in the object graph sent between the server and the client, network saturation and/or load-balancing information, and user hints.

[0024] The application can start with an initial usage pattern derived from application code analysis or from an empty set. As the web client **20** sends requests to the web server **15** via **25**, and the web components **10** process the requests, the objects requested, created, modified, and destroyed by the web components **10** are monitored. Also, all the attributes accessed and all the remote calls made are monitored. This allows a graph of objects and attributes to be constructed. If learning is based on URL, for example, a graph of objects and attributes per URL is constructed. The next time the URL is accessed, objects and attributes are pre-fetched from the application server **35** according to graph of objects and attributes for that URL. The client runtime **65** sends the list of objects and attributes to be pre-fetched to a server runtime **70** that actually retrieves the data.

[0025] The server runtime **70** coordinates synchronization of the client cache **60** with the persistent objects **40**. In order to reduce the number of client-server roundtrips, the client cache **60** is synchronized after each remote method call based on the history of the client-side (proxy) objects modified by the previous method invocations and/or user hints. For each data roundtrip, the optimal amount of data to send from the web server **15** to the application server **35** is determined. This optimal amount of data is based on the attributes of the cached objects changed by the user. Also, the optimal amount of data to be sent from the application server **35** to the web server **15** is determined. This optimal amount of data is based on the attributes of the cached objects changed by the user or application business logic.

[0026] The present invention provides general advantages in that an application that does not originally include a facility for learning data usage pattern can be instrumented to learn data usage pattern. The learned data usage pattern can be used advantageously to determine the optimal amount of data to transfer between the client and server, reducing network traffic and memory space used by the client cache. The present invention can also facilitate the development and optimization of business process models, and system

test frameworks. The optimization can be done transparently, *i.e.*, without changing the application semantic.

[0027] While the present invention has been described with respect to a limited number of embodiments, those skilled in the art, having benefit of this disclosure, will appreciate that other embodiments can be devised which do not depart from the scope of the present invention as disclosed herein. Accordingly, the scope of the present invention should be limited only by the attached claims.